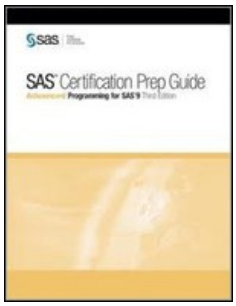


SAS Certification Prep Guide: Advanced Programming for SAS 9, Third Edition

by SAS Institute

Chapter 12: Storing Macro Programs

Overview

Introduction

One of the most useful aspects of macro programming is the ability to reuse your macro programs. In "Creating and Using Macro Programs" on page 390, you learned that compiled macros are stored in a temporary SAS catalog by default and are available for execution anytime during the current SAS session. You also learned that macros stored in this temporary SAS catalog are deleted at the end of the SAS session.

You might want to store your macros *permanently* so that you can reuse them in later SAS sessions or share them with others. There are several ways of storing your macro programs permanently and of making them accessible during a SAS session. The methods that you will learn in this chapter are

- the %INCLUDE statement
- the autocall macro facility
- permanently stored compiled macros.

Objectives

In this chapter, you learn to

- use the %INCLUDE statement to make macros available to a SAS program
- use the autocall macro facility to make macros available to a SAS program
- use SAS system options with the autocall facility
- create and use permanently stored compiled macros.

Prerequisites

Before beginning this chapter, you should complete the following chapters:

- "Performing Queries Using PROC SQL" on page 4
- "Introducing Macro Variables" on page 304
- "Processing Macro Variables at Execution Time" on page 344
- "Creating and Using Macro Programs" on page 390.

Understanding Session-Compiled Macros

In "Creating and Using Macro Programs" on page 390, you learned that you can submit a macro definition in order to compile a macro. For example, when you submit the macro definition shown here, the macro processor compiles the macro *Prtlast*.

```
%macro prtlast;
   %if &syslast ne _NULL_ %then %do;
      proc print data=&syslast(obs=5);
         title "Listing of &syslast data set";
      run;
   %end;
   %else
      %put No data set has been created yet.;
%mend;
```

By default, the *Prtlast* macro is stored in a temporary SAS catalog as *Work.Sasmacr.Prtlast.Macro*. Macros that are stored in this temporary SAS catalog are known as *session-compiled macros*. Once a macro has been compiled, it can be

invoked from a SAS program as shown here:

```
proc sort data=sasuser.courses out=bydays;
   by days;
run;

%prtlast
```

Session-compiled macros are available for execution during the SAS session in which they are compiled. They are deleted at the end of the session. But suppose you want to save your macros so that they are not deleted at the end of the SAS session. The rest of this chapter looks at methods of storing macros permanently.

Storing Macro Definitions in External Files

Overview

One way to store macro programs permanently is to save them to an external file. You can then use the %INCLUDE statement to insert the statements that are stored in the external file into a program. If the external file contains a macro definition, the macro is compiled when the %INCLUDE statement is submitted. Then the macro can be called again later in the same program, or anytime later in the current SAS session.

General form, %INCLUDE statement:

%INCLUDE *file-specification* </SOURCE2>;

where

file-specification

describes the location of the file that contains the SAS code to be inserted.

SOURCE2

causes the SAS statements that are inserted into the program to be displayed in the SAS log. If SOURCE2 is not specified in the %INCLUDE statement, then the setting of the *SAS system option* SOURCE2 controls whether the inserted code is displayed.

By storing your macro program externally and using the %INCLUDE statement, you gain several advantages over using session-compiled macros.

- The source code for the macro definition does not need to be part of your program.
- A single copy of a macro definition can be shared by many programs.
- Macro definitions in external files are easily viewed and edited with any text editor.
- No special SAS system options are required in order to access a macro definition that is stored in an external file.

Example

You can compile a macro by using the %INCLUDE statement to insert its definition into a program. Then you can call the macro in order to execute it.

Suppose the following macro definition is stored in the external file

*C:**sasfiles**prtlast.sas*\

```
%macro prtlast;
   %if &syslast ne _NULL_ %then %do;
      proc print data=&syslast (obs=5);
         title "Listing of &syslast data set";
      run;
   %end;
   %else
```

```
      %put No data set has been created yet.;
%mend;
```

You could submit the following code to access, compile, and execute the *Prtlast* macro. The PROC SORT step is included in this example in order to create a data set that the *Prtlast* macro can print.

```
%include 'c:\sasfiles\prtlast.sas' /source2;

proc sort data=sasuser.courses out=bydays;
   by days;
run;

%prtlast
```

Note The location and names of external files will be specific to your operating environment.

The following messages are written to the SAS log when this code is submitted. Notice that the macro definition is written to the log because **SOURCE2** was specified in the %INCLUDE statement.

Table 12.1: SAS Log

```
NOTE: %INCLUDE (level 1) file prtlast.sas is file
              C:\sasfiles\prtlast.sas.
31   +%macro prtlast;
32   +   %if &syslast ne _NULL_ %then %do;
33   +      proc print data=&syslast(obs=5);
34   +      title "Listing of &syslast data set";
35   +      run;
36   +   %end;
37   +   %else
38   +      %put No data set has been created yet.;
39   +%mend;
NOTE: %INCLUDE (level 1) ending.
40
41   proc sort data=sasuser.courses out=bydays;
42      by days;
43   run;

NOTE: There were 6 observations read from the dataset
      SASUSER.COURSES.
NOTE: The data set WORK.BYDAYS has 6 observations and
      4 variables.
NOTE: PROCEDURE SORT used:
      real time           0.04 seconds
      cpu time            0.04 seconds

44
45   %prtlast
NOTE: There were 5 observations read from the dataset
      WORK.BYDAYS.
NOTE: PROCEDURE PRINT used:
      real time           1.07 seconds
      cpu time            0.26 seconds
```

Here is the output that the code generates.

Listing of WORK.BYDAYS data set

Obs	Course_Code	Course_Title	Days	Fee
1	C004	Database Design	2	$375
2	C005	Artificial Intelligence	2	$400
3	C001	Basic Telecommunications	3	$795
4	C003	Local Area Networks	3	$650
5	C002	Structured Query Language	4	$1150

Storing Macro Definitions in Catalog SOURCE Entries

Overview

Another way of permanently storing macros is to store a macro definition in a *SOURCE entry* in a SAS catalog. If you decide to store your macro programs in a SAS catalog, you must store each macro program in a separate SOURCE entry. It is a good idea to give each SOURCE entry the *same name* as the macro program that is stored within it. For example, a macro named *Printit* would be stored in a SOURCE entry that is also named *Printit*.

Note SAS catalogs are members of SAS libraries that store program source code and other types of content.

To store a macro definition as a SOURCE entry in a SAS catalog, you use the Save As Object window.

Example

To save the *Printit* macro definition to the *Sasuser.Mymacs* catalog, perform these steps:

1. Select **File** ⇨ **Save As Object**. In the Save As Object window, select the **Sasuser** library.
2. If the *Sasuser.Mymacs* catalog does not already exist, you need to create it. You can either select the **Create New Catalog** icon or right-click the Save As Object window and select **New** in order to open the New Catalog window. Enter **Mymacs** as the name for the new catalog and click **OK**.
3. Enter **Printit** in the Entry Name field. Make sure that the Entry Type is set to **SOURCE entry (SOURCE)**, and click **Save**.

Tip If you use the Program Editor, you could also use the SAVE command to save your macro definition as a catalog SOURCE entry. To use the SAVE command, you enter save *libref.catalog.entry.source* in the command line where *libref.catalog.entry* is the libref, the catalog name, and the entry name.

The CATALOG Procedure

If you store your macros in a SAS catalog, you might want to view the contents of a particular catalog to see the macros you have stored there. You can use the Explorer window to view the contents of a SAS catalog by navigating to the catalog and double clicking it. You can also use the CATALOG procedure to list the contents of a SAS catalog. The CONTENTS statement of the CATALOG procedure lists the contents of a catalog in the procedure output.

General form, CATALOG procedure with CONTENTS statement:

```
PROC CATALOG CATALOG=libref.catalog;
     CONTENTS;
QUIT;
```

where

libref.catalog

is a valid two-level catalog name.

Note CAT= is an alias for CATALOG.

Example

You can use PROC CATALOG to view all of the macros that are stored in the temporary *Work.Sasmacr* catalog, as follows:

```
proc catalog cat=work.sasmacr;
   contents;
   title "Default Storage of SAS Macros";
quit;
```

This PROC CATALOG step produces results that are similar to the output shown below. The macros that are actually

listed will be the macros that have been compiled during the current SAS session.

Default Storage of SAS Macros

Contents of Catalog WORK.SASMACR

#	Name	Type	Create Date	Modified Date	Description
1	PRTLAST	MACRO	20Apr11:14:22:34	20Apr11:14:22:34	
2	SORTLAST2	MACRO	20Apr11:14:32:22	20Apr11:14:32:22	
3	TRIM	MACRO	20Apr11:15:01:19	20Apr11:15:01:19	

PROC CATALOG can display the names and attributes of compiled macros, but the macro definition it self cannot be viewed.

The CATALOG Access Method

If you store a macro definition in a SOURCE entry of a SAS catalog, you can use the CATALOG access method in a FILENAME statement in conjunction with the %INCLUDE statement to insert the macro definition into a SAS program.

General form, CATALOG access method to reference a single SOURCE entry:

```
FILENAME fileref
     CATALOG 'libref.catalog.entry-name.entry-type';
%INCLUDE fileref;
```

where

fileref

is a valid fileref.

libref.catalog.entry-name.entry-type

is a four-level SAS catalog entry name.

entry-type

is SOURCE.

Example

Suppose you have stored the following macro definition as a SOURCE entry in the SAS catalog *Sasuser.Mymacs:*

```
%macro prtlast;
   %if &syslast ne _NULL_ %then %do;
      proc print data=&syslast(obs=5);
         title "Listing of &syslast data set";
      run;
   %end
   %else
      %put No data set has been created yet.;
%mend;
```

You can use the CATALOG access method along with the %INCLUDE statement to compile the macro *Prtlast*. Then you can reference the macro later in the program.

```
filename prtlast catalog 'sasuser.mymacs.prtlast.source';
%include prtlast;
proc sort data=sasuser.courses out=bydays;
   by days;
run;
%prtlast
```

You can also use the CATALOG access method to reference multiple SOURCE entries as long as the entries are stored in

the same SAS catalog.

General form, CATALOG access method to reference multiple SOURCE entries:

FILENAME *fileref* **CATALOG** *'libref.catalog';*

%INCLUDE *fileref(entry-1);*

%INCLUDE *fileref(entry-2);*

where

fileref

is a valid fileref.

libref.catalog

is a two-level catalog name.

entry-1 and *entry-2*

are names of SOURCE entries in *library.catalog*.

Example

Suppose you have two macros, named *Prtlast* and *Sortlast*, that are stored in a SAS catalog.

Catalog Entry: **Sasuser.Mymacs.Prtlast.Source**

```
%macro prtlast;
   %if &syslast ne _NULL_ %then %do;
      proc print data=&syslast(obs=5);
      title "Listing of &syslast data set";
      run;
   %end;
   %else
     %put No data set has been created yet.;
%mend;
```

Catalog Entry: **Sasuser.Mymacs.Sortlast.Source**

```
%macro sortlast(sortby);
   %if &syslast ne _NULL_ %then %do;
      proc sort data=&syslast out=sorted;
         by &sortby;
      run;
   %end;
   %else
     %put No data set has been created yet.;
%mend;
```

You can use the CATALOG access method in conjunction with the %INCLUDE statement to compile both macros. Then you can call the macros later in the program. In this example, assume that the macros have the same names as the SOURCE entries in which they are stored:

```
filename prtsort catalog 'sasuser.mymacs';
%include prtsort(prtlast) / source2;
%include prtsort(sortlast) / source2;

data current(keep=student_name course_title begin_date location);
   set sasuser.all;
   if year(begin_date)=2001;
   diff=year(today())-year(begin_date);
   begin_date=begin_date+(365*diff);
run;
```

```
%sortlast(begin_date)
%prtlast
```

This code produces the following output:

Listing of WORK.SORTED data set

Obs	Student_Name	Location	Begin_Date	Course_Title
1	Bills, Ms. Paulette	Boston	06JAN2011	Local Area Networks
2	Chevarley, Ms. Arlene	Boston	06JAN2011	Local Area Networks
3	Clough, Ms. Patti	Boston	06JAN2011	Local Area Networks
4	Crace, Mr. Ron	Boston	06JAN2011	Local Area Networks
5	Davis, Mr. Bruce	Boston	06JAN2011	Local Area Networks

Using the Autocall Facility

Overview

You can make macros accessible to your SAS session or program by using the *autocall facility* to search predefined source libraries for macro definitions. These predefined source libraries are known as *autocall libraries*. You can store your macro definitions permanently in an autocall library, and you can set up multiple autocall libraries.

When you use this approach, you do not need to compile the macro in order to make it available for execution. That is, if the macro definition is stored in an autocall library, then you do not need to submit or include the macro definition before you submit a call to the macro.

Suppose you have stored a file that contains a macro definition in your autocall library. When you submit a call to that macro

- the macro processor searches the autocall library for the macro
- the macro is compiled and stored as it would be if you had submitted it (that is, the compiled macro is stored in the default location of *Work.Sasmacr)*
- the macro is executed.

Once it has been compiled, the macro can be executed as needed throughout the same SAS session. At the end of the SAS session, the compiled macro is deleted from the *Work.Sasmacr* catalog, but the source code remains in the autocall library.

Creating an Autocall Library

An autocall library can be either

- a directory that contains source files
- a partitioned data set (PDS)
- a SAS catalog.

The method for creating an autocall library depends on the operating environment that you are using.

To create an autocall library in a directory-based operating system, such as Windows or UNIX, create a directory in which to store macro definitions. Each macro definition in this directory will be a separate file that has the extension *.sas* and that has the same name as the macro that it contains.

Example

Suppose you want to save the macro *Prtlast* in an autocall library. In a directory-based operating system, the first step is to create a directory that will hold your macro source files. You can use the Save As window to create the directory, and to save the macro definition in that directory. With the *Prtlast* definition in an active code editing window, select **File** ⇨ **Save**

As. In the Save As window, navigate to the location where you want to create your autocall library. Select New Folder, enter the directory name, and click OK. Then enter Prtlast as the filename, make sure the file type is .sas, and click Save.

Tip You could also use the FILE command to save your macro definition in an autocall library. To use the FILE command, you enter

`file '<path>external-file-name'` in the command line.

Default Autocall Library

SAS provides several macros in a default autocall library for you. Some of the macros in the autocall library that SAS provides are listed here.

Macro Syntax	Purpose
%LOWCASE (*argument*)	converts letters in its argument from uppercase to lowercase
%QLOWCASE (*argument*)	converts letters in its argument from uppercase to lowercase, and returns a result that masks special characters and mnemonic operators
%LEFT(*argument*)	removes leading blanks from the argument
%TRIM(*argument*)	removes trailing blanks from the argument
%CMPRES(*argument*)	removes multiple blanks from the argument
%DATATYP (*argument*)	returns the string NUMERIC or CHAR, depending on whether the argument is an integer or a character string

You might be familiar with SAS functions such as TRIM and LEFT. The macros that SAS supplies look like macro functions, but they are in fact macros. One of the useful things about these macros is that in addition to using them in your SAS programs, you can see their source code.

Example

The macro definition for the *Lowcase* macro is shown below. Notice that the comments that are included in this macro provide information about using the macro. All of the macros that SAS provides in the autocall library include explanatory comments so that they will be easy for you to understand and use.

```
%macro lowcase(string);
%*****************************************************;
%*                                                   *;
%* MACRO: LOWCASE                                    *;
%*                                                   *;
%* USAGE: 1) %lowcase(argument)                      *;
%*                                                   *;
%* DESCRIPTION:                                      *;
%*   This macro returns the argument passed to       *;
%*   it unchanged except that all upper-case         *;
%*   alphabetic characters are changed to their      *;
%*   lower-case equivalents.                         *;
%*                                                   *;
%* E.g.: %let macvar=%lowcase(SAS Institute Inc.);   *;
%* The variable macvar gets the value                *;
%*   "sas institute inc."                            *;
%* NOTES:                                            *;
%*   Although the argument to the %UPCASE macro      *;
%*   function may contain commas, the argument to    *;
%*   %LOWCASE may not, unless they are quoted.       *;
%*   Because %LOWCASE is a macro, not a function,    *;
%*   it interprets a comma as the end of a parameter. *;
%*****************************************************;
%sysfunc(lowcase(%nrbquote(&string)))
%mend;
```

Accessing Autocall Macros

Remember that an autocall library is either a SAS catalog, an external directory, or a partitioned data set. This is true both for the default autocall library that SAS supplies and for autocall libraries that you create.

In order to access a macro definition that is stored in an autocall library, you must use two SAS system options, as follows:

- The MAUTOSOURCE system option must be specified.
- The SASAUTOS= system option must be set to identify the location of the autocall library or libraries.

Both the MAUTOSOURCE and SASAUTOS= system options can be set either at SAS invocation or with an OPTIONS statement during program execution.

The MAUTOSOURCE system option controls whether the autocall facility is available.

General form, MAUTOSOURCE system option:

OPTIONS MAUTOSOURCE | NOMAUTOSOURCE;

where

MAUTOSOURCE

is the default setting, and specifies that the autocall facility is available.

NOMAUTOSOURCE

specifies that the autocall facility is not available.

The SASAUTOS= system option controls where the macro facility looks for autocall macros.

General form, SASAUTOS= system option:

OPTIONS SASAUTOS=*library-1*;

OPTIONS SASAUTOS=*(library-l,…, library-n);*

where

the values of *library-1* through *library-n*

are references to source libraries that contain macro definitions. To specify a source library you can

- use a fileref to refer to its location
- specify the pathname (enclosed in quotation marks) for the library.

Unless your system administrator has changed the default value for the SASAUTOS=system option, its value is the fileref *Sasautos*, and that filerefpoints to the location where the default autocall library was created during installation. The *Sasautos* fileref can refer to multiple locations that are concatenated.

Generally, it is a good idea to concatenate any autocall libraries that you create yourself with the default autocall library in the value of the SASAUTOS=system option. Otherwise, the new autocall library will replace the default or existing libraries in the value of SASAUTOS=, and the autocall facility will have access to only the new autocall library.

Example

Suppose you want to access the *Prtlast* macro, which is stored in the autocall library *C:\Mysasfiles*. You also want to make sure that the default autocall library (which the fileref *Sasautos* points to) is still available to the autocall facility. You would submit the following code:

```
options mautosource sasautos=('c:\mysasfiles', sasautos);
%prtlast
```

Note The *MAUTOLOCDISPLAY option* is a Boolean option that causes a note tobe issued to the SAS log indicating where the source code was obtained to compile an autocall macro. The note is similar to the information displayed when using the MLOGIC option. The default setting of this option is NOMAUTOLOCDISPLAY.

When the autocall facility is in effect, if you invoke a macro that has not been previously compiled, the macro facility automatically

1. searches the autocall library (or each autocall library in turn if multiple libraries are identified in the SASAUTOS=system option) for a member that has the same name as the invoked macro
2. brings the source statements into the current SAS session if the member is found
3. issues an error message if the member is not found
4. submits all statements in the member in order to compile the macro
5. stores the compiled macro in the temporary catalog *Work.Sasmacr*
6. calls the macro.

The autocall facility does not search for a macro in the autocall library if the macro has already been compiled during the current SAS session. In that case, the session-compiled macro is executed.

Using Stored Compiled Macros

The Stored Compiled Macro Facility

Remember that when a macro is compiled, it is stored in the temporary SAS catalog *Work.Sasmacr* by default. You can also store compiled macros in a permanent SAS catalog. Then you can use the *Stored Compiled Macro Facility* to access permanent SAS catalogs that contain compiled macros.

There are several advantages to using stored compiled macros:

- SAS does not need to compile a macro definition when a macro call is made.
- Session-compiled macros and the autocall facility are also available in the same session.
- Users cannot modify compiled macros.

Two SAS system options affect stored compiled macros: MSTORED and SASMSTORE-The MSTORED system option controls whether the Stored Compiled Macro Facility is available.

General form, MSTORED system option:

OPTIONS MSTORED | NOMSTORED;

where

NOMSTORED

is the default setting, and specifies that the Stored Compiled Macro Facility does not search for compiled macros.

MSTORED

specifies that the Stored Compiled Macro Facility searches for stored compiled macros in a catalog in the SAS library that is referenced by the SASMSTORE= option.

The SASMSTORE=system option controls where the macro facility looks for stored compiled macros.

General form, SASMSTORE= system option:

OPTIONS SASMSTORE=*libref*;

where

libref

specifies the libref of a SAS library that contains, or will contain, a catalog of stored compiled SAS macros. This librefcannot be *Work*.

The MSTORED and SASMSTORE=system options can be set either at SAS invocation or with an OPTIONS statement during program execution.

Creating a Stored Compiled Macro

To create a permanently stored compiled macro, you must

1. assign a libref to the SAS library in which the compiled macro will be stored
2. set the system options MSTORED and SASMSTORE=*libref*
3. use the STORE option in the %MACRO statement when you submit the macro definition.

General form, macro definition with STORE option:,

```
%MACRO macro-name <(parameter-list)> /STORE
        <DES= 'description>;
        text
%MEND <macro-name>;
```

where

description

is an optional 156-character description that appears in the catalog directory.

macro-name

names the macro.

parameter-list

names one or more local macro variables whose values you specify when you invoke the macro.

text

can be

- constant text, possibly including SAS data set names, SAS variable names, or SAS statements
- macro variables, macro functions, or macro program statements
- any combination of the above.

There are several restrictions on stored compiled macros.

- *Sasmacr* is the *only* catalog in which compiled macros can be stored. You can create a catalog named *Sasmacr* in any SAS library. You should not rename this catalog or its entries.
- You *cannot* copy stored compiled macros across operating systems. You must copy the source program and re-create the stored compiled macro.

- The source *cannot* be re-created from the compiled macro. You should retain the original source program. For convenience, you can store the source program in an autocall library. Alternatively, you can store the source program as a source entry in the same catalog as the compiled macro.

Using the SOURCE Option

An alternative to saving your source program separately from the stored compiled macro is to use the SOURCE option in the %MACRO statement to combine and store the source of the compiled macro with the compiled macro code. The SOURCE option requires that the STORE option and the MSTORED option be set. The %MACRO statement below shows the correct syntax for using the SOURCE option.

```
%macro macro-name<(parameter list)> /STORE SOURCE;
```

The source code that is saved by the SOURCE option begins with the %MACRO keyword and ends with the semicolon following the %MEND statement.

Tip The SOURCE option cannot be used on nested macro definitions.

Example

Suppose you want to store the *Words* macro in compiled form in a SAS library. This example shows the macro definition for *Words*. The macro takes a text string, divides it into words, and creates a series of macro variables to store each word.

Notice that both the STORE option and the SOURCE option are used in the macro definition so that *Words* will be permanently stored as a compiled macro and the macro source code will be stored with it, as follows:

```
libname macrolib 'c:\storedlib';
options mstored sasmstore=macrolib;

%macro words(text,root=w,delim=%str( ))/store source;
   %local i word;
   %let i=1;
   %let word=%scan(&text,&i,&delim);
   %do %while (&word ne);
      %global &root&i;
      %let &root&i=&word;
      %let i=%eval(&i+1);
      %let word=%scan(&text,&i,&delim);
   %end;
   %global &root.num;
   %let &root.num=%eval(&i-1);
%mend words;
```

If the *Sasmacr* catalog does not exist in the *Macrolib* library, it is automatically created. You can list the contents of the *Macrolib.Sasmacr* catalog to verify that the compiled macro was created, as follows:

```
proc catalog cat=macrolib.sasmacr;
   contents;
   title "Stored Compiled Macros";
quit;
```

Here is the output from the PROC CATALOG step if no other compiled macros are stored in *Macrolib.Sasmacr*.

Stored Compiled Macros

Contents of Catalog MACROLIB.SASMACR

#	Name	Type	Create Date	Modified Gate	Description
1	WORDS	MACRO	20Apr11:15:35:55	20Apr11:15:35:55	

Accessing Stored Compiled Macros

In order to access a stored compiled macro, you must

1. assign a libref to the SAS library that contains a *Sasmacr* catalog in which the macro was stored
2. set the system options MSTORED and SASMSTORE=*libref*

3. call the macro

Example

The following program calls the *Words* macro. Assume that the *Words* macro was compiled and stored in an earlier SAS session.

```
libname macrolib 'c:\storedlib';
options mstored sasmstore=macrolib;

%words(This is a test)
%put Number of Words (wnum): &wnum;
%put Word Number 1 (w1): &w1;
%put Word Number 2 (w2): &w2;
%put Word Number 3 (w3): &w3;
%put Word Number 4 (w4): &w4;
```

Here is a portion of the messages that are written to the SAS log when this code is submitted.

Table 12.2: SAS Log

```
9     libname macrolib 'c:\storedlib';
NOTE:  Libref MACROLIB was successfully assigned as follows:
       Engine: V9
       Physical Name: c:\storedlib
10        options mstored sasmstore=macrolib;
11
12        %words(This is a test)
13        %put Number of Words (wnum): &wnum;
Number of Words (wnum): 4
14        %put Word Number 1 (w1): &w1;
Word Number 1 (w1): This
15        %put Word Number 2 (w2): &w2;
Word Number 2 (w2): is
16        %put Word Number 3 (w3): &w3;
Word Number 3 (w3): a
17        %put Word Number 4 (w4): &w4;
Word Number 4 (w4): test
```

Accessing Stored Macro Code

If you use the SOURCE option of the %MACRO statement to store your macro source code along with the stored compiled macro, you can use the %COPY statement to access the stored source code.

General form, %COPY statement:

%COPY *macro-name* **/SOURCE** *<other option(s)>;*

where

macro-name

is the name of the macro whose source code will be accessed.

SOURCE

specifies that the source code of the macro will be copied to the output destination. If no output destination is specified, the source is written to the SAS log.

other options

include the following options:

- LIBRARY=specifies the libref of a SAS library that contains a catalog of stored compiled SAS macros. If no library is

specified, the libref specified by the SASMSTORE=option is used. The libref cannot be *Work*.

- OUTFILE=specifies the output destination of the %COPY statement. The value can be a fileref or an external file.

Example

Suppose you submitted the program below to create a stored compiled macro named *Words*.

```
libname macrolib 'c:\storedlib';
options mstored sasmstore=macrolib;

%macro words(text,root=w,delim=%str( ))/store source;
   %local i word;
   %let i=1;
   %let word=%scan(&text,&i,&delim);
   %do %while (&word ne);
      %global &root&i;
      %let &root&i=&word;
      %let i=%eval(&i+1);
      %let word=%scan(&text,&i,&delim);
      %end;
   %global &root.num;
   %let &root.num=%eval(&i-1);
%mend words;
```

The %COPY statement writes the source code for the *Words* macro to the SAS log. Here is an example:

```
%copy words/source;
```

The partial SAS log below shows the source code of the *Words* macro.

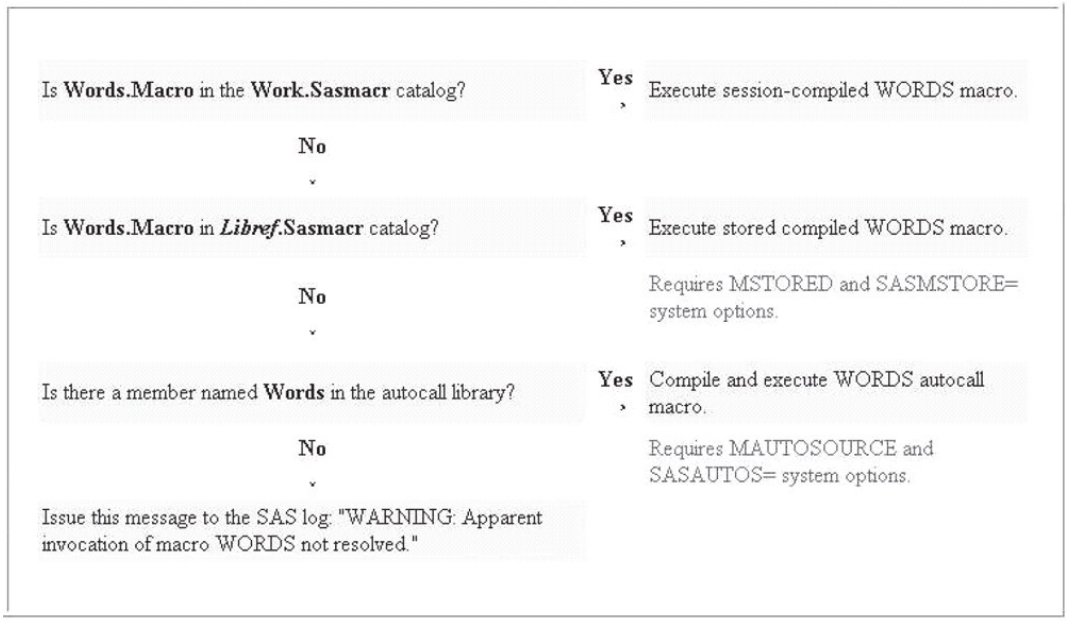
Table 12.3: SAS Log

```
17     %copy words/source;
%macro words(text,root=w,delim=%str( ))/store source;

       %local i word;
       %let i=1;
       %let word=%scan(&text,&i,&delim);
       %do %while (&word ne);
          %global &root&i;
          %let &root&i=&word;
          %let i=%eval(&i+1);
          %let word=%scan(&text,&i,&delim);
       %end;
       %global &root.num;
       %let &root.num=%eval(&i-1);
%mend words;
```

The Stored Compiled Macro Facility can be used in conjunction with the Autocall Facility and with session-compiled macros. When you submit a macro call such as **`%words`**, the macro processor searches for the macro *Words* as

1. an entry named *Words.Macro* in the temporary *Work.Sasmacr* catalog.
2. an entry named *Words.Macro* in the *Libref.Sasmacr* catalog. The MSTORED option must be specified, and the SASMSTORE= option must have a value *Libref*.
3. an autocall library member named *Words* that contains the macro definition for the macro *Words*. The MAUTOSOURCE option must be specified, and the value of the SASAUTOS= option must point to the autocall library.



Summary

Contents

This section contains the following topics.

Text Summary

Understanding Session-Compiled Macros

You can make a macro available to your SAS session by submitting the macro definition before calling the macro. This creates a session-compiled macro. Session-compiled macros are deleted from the temporary SAS catalog *Work.Sasmacr* at the end of the session.

Storing Macro Definitions in External Files

One way to store your macro definitions permanently is to save them in external files. You can make a macro definition that is stored in an external file accessible to your SAS programs by using the %INCLUDE statement.

Storing Macro Definitions in Catalog SOURCE Entries

You can also store your macro definitions permanently as SOURCE entries in SAS catalogs. You can use the catalog access method to make these macros accessible to your SAS programs. The PROC CATALOG statement enables you to view a list of the contents of a SAS catalog.

Using the Autocall Facility

You can permanently store macro definitions in source libraries called autocall libraries. SAS provides several macro definitions for you in a default autocall library. You can concatenate multiple autocall libraries. To access macros that are stored in an autocall library, you specify the SASAUTOS= and MAUTOSOURCE system options.

Using Stored Compiled Macros

Another efficient way to make macros available to a program is to store them in compiled form in a SAS library. To store a compiled macro permanently, you must set two system options, MSTORED and SASMSTORE" Then you submit one or

more macro definitions, using the STORE option in the %MACRO statement. The compiled macro is stored as a catalog entry in *Libref.Sasmacr*. The source program is not stored as part of the compiled macro. You should always maintain the original source program for each macro definition in case you need to redefine the macro. You can use the SOURCE option in the %MACRO statement to store the macro source code with the compiled macro. If you use the SOURCE option in the %MACRO statement, you can use the %COPY statement to access the macro source code later.

Syntax

```
%INCLUDE file-specification </SOURCE2>;
FILENAME fileref
     CATALOG 'libref.catalog.entry-name.entry-type';
%INCLUDE fileref;
FILENAME fileref CATALOG 'libref.catalog';
%INCLUDE fileref(entry-1);
%INCLUDE fileref(entry-2);
PROC CATALOG CATALOG=libref.catalog;
     CONTENTS;
QUIT;
OPTIONS MAUTOSOURCE | NOMAUTOSOURCE;
OPTIONS SASAVTOS=library-1;
OPTIONS SASAVTOS=(library-1,…,library-n);
OPTIONS MSTORED | NOMSTORED;
OPTIONS SASMSTORE=libref;
%MACRO macro-name<(parameters-list)> /STORE SOURCE
    <DES='description';
    text
%MEND <macro-name>;
```

Sample Programs

Compiling an Externally Stored Macro Definition with the %INCLUDE Statement

```
%include 'c:\sasfiles\prtlast.sas' / source2;

proc sort data=sasuser.courses out=bydays;
   by days;
run;

%prtlast
```

Listing the Contents of a Catalog

```
proc catalog cat=work.sasmacr;
   contents;
   title "Default Storage of SAS Macros";
quit;
```

Using the Catalog Access Method

```
filename prtlast catalog 'sasuser.mymacs.prtlast.source';
%include prtlast;
proc sort data=sasuser.courses out=bydays;
   by days;
run;
%prtlast
```

Accessing an Autocall Macro

```
options mautosource sasautos=('c:\mysasfiles',sasautos);
%prtlast
```

Creating a Stored Compiled Macro

```
libname macrolib 'c:\storedlib';
options mstored sasmstore=macrolib;

%macro words(text,root=w,delim=%str( ))/store;
   %local i word;
   %let i=1;
   %let word=%scan(&text,&i,&delim);
   %do %while (&word ne);
```

```
      %global &root&i;
      %let &root&i=&word;
      %let i=%eval(&i+1);
      %let word=%scan(&text,&i,&delim);
   %end;
   %global &root.num;
   %let &root.num=%eval(&i-1);
%mend words;
```

Points to Remember

- You can make macros available to your programs in four ways: as session-compiled macros, with a %INCLUDE statement, through the autocall facility, or as stored compiled macros.
- If you use the autocall facility, you must specify the MAUTOSOURCE and SASAUTOS= system options.
- If you use the stored compiled macro facility, you must specify the MSTORED and SASMSTORE= system options.
- The point at which macro compilation occurs depends on which method you use to

  access the macro.

Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

1. The %INCLUDE statement ?

a. can be used to insert the contents of an external file into a program.

b. will cause a macro definition that is stored in an external file to be compiled when the contents of that file are inserted into a program and submitted.

c. can be specified with the SOURCE2 option in order to write the contents of the external file that is inserted into a program to the SAS log.

d. all of the above

2. If you store a macro definition in a SAS catalog SOURCE entry ?

a. the macro definition can be submitted for compilation by using the FILENAME and %INCLUDE statements.

b. you can use the PROC CATALOG statement to compile the macro.

c. the SOURCE entry will be deleted at the end of the session.

d. you do not need to compile the macro before you invoke it in a program.

3. Which of the following programs correctly sets the appropriate system options and calls the macro *Prtlast*? Assume that *Prtlast* is stored in an autocall library as a text file and that it has not been compiled during the current SAS session. ?

a.
```
a. libname mylib 'c:\mylib';
   filename macsrc 'mylib.macsrc';
   options mautosource sasautos=(macsrc, sasautos);
   %prtlast
```

b.
```
b.  libname mylib 'c:\mylib';
    filename macsrc catalog 'mylib.macsrc';
    %prtlast
```

c.
```
c.  filename mylib 'c:\mylib';
    options mautosource sasautos=(sasautos,mylib);
    %prtlast
```

d. d.
```
libname mylib 'c:\mylib';
options mautosource sasautos=mylib;
%prtlast
```

4. If you use the Stored Compiled Macro Facility, ?

a. the macro processor does not compile a macro every time it is used.

b. the only compiled macros that the Stored Compiled Macro Facility can access are those that are stored in the *Sasmacr* catalog.

c. you need to specify the MSTORED and SASMSTORE= system options.

d. all of the above

5. Which of the following correctly creates a permanently stored compiled macro? ?

a. a.
```
libname macrolib 'c:\mylib';
options sasmstore;
%macro prtlast; / store
   proc print data=&syslast (obs=5);
      title "Listing of &syslast data set";
   run;
%mend;
```

b. b.
```
libname macrolib 'c:\mylib';
options mstored sasmstore=macrolib;
%macro prtlast / store;
  proc print data=&syslast (obs=5);
     title "Listing of &syslast data set";
  run;
%mend;
```

c. c.
```
libname macrolib 'c:\mylib';
options mstored sasmstore=macrolib;
%macro prtlast;
  proc print data=&syslast (obs=5);
     title "Listing of &syslast data set";
  run;
%mend;
```

d. d.
```
libname macrolib 'c:\mylib';
%macro prtlast / store;
  proc print data=&syslast (obs=5);
     title "Listing of &syslast data set";
  run;
%mend;
```

6. When you submit the following code, what happens? ?

```
%macro prtlast;
   proc print data=&syslast (obs=5);
      title "Listing of &syslast data set";
   run;
%mend;
```

a. A session-compiled macro named *Prtlast* is stored in *Work.Sasmacr*.

b. A macro named *Prtlast* is stored in the autocall library.

c. The *Prtlast* macro is stored as a stored compiled macro.

d. The *Prtlast* macro is stored as a SOURCE entry in a permanent SAS catalog.

7. Why would you want to store your macros in external files? ?

a. You could easily share your macros with others.

b. You could edit your macros with any text editor.

c. Your macros would be available for use in later SAS sessions.

d. all of the above

8. What will the following PROC CATALOG step do? ?

```
proc catalog cat=mylib.sasmacr;
   contents;
quit;
```

a. Copy the contents of the *Sasmacr* catalog to a temporary data set.

b. List the contents of the *Sasmacr* catalog as output.

c. Copy the contents of the output window to the *Sasmacr* catalog.

d. none of the above

9. Which of the following is not true about stored compiled macros? ?

a. Because these stored macros are compiled, you should save and maintain the source for the macro definitions in a different location.

b. The Stored Compiled Macro Facility compiles and saves compiled macros in a permanent catalog, in a library that you specify.

c. You do not need to specify any system options in order to use the Stored Compiled Macro Facility.

d. You cannot move a stored compiled macro to another operating system.

10. Which of the following is not true? ?

a. The autocall macro facility stores compiled SAS macros in a collection of external files called an autocall library.

b. Autocall libraries can be concatenated together.

c. One disadvantage of the autocall facility is that the first time you call an autocall macro in a SAS session, the macro processor must use system resources to compile it.

d. The autocall facility can be used in conjunction with the Stored Compiled Macro Facility.

Answers

1. Correct answer: d

The %INCLUDE statement can be used to insert the contents of an external file into a SAS program. If a macro definition is stored in an external file, the %INCLUDE statement causes the macro definition to be compiled when it is inserted into the SAS program. The contents of the macro definition will be written to the SAS log only if the SOURCE2 option is specified.

2. Correct answer: a

When a macro definition is stored as a catalog SOURCE entry, you must compile it before you can call it from a SAS program. You compile a macro that is stored as a catalog SOURCE entry by using the CATALOG access method. This creates a session-compiled macro that will be deleted at the end of the SAS session. The PROC CATALOG statement enables you to view a list of the contents of a SAS catalog.

3. Correct answer: c

To call a macro that is stored in an autocall library, you must specify both the MAUTOSOURCE system options and the SASAUTOS= system option. The SASAUTOS= system option can be set to include multiple pathnames or filerefs. Once these two system options are set, you can call the macro by preceding the macro name with a percent sign.

4. Correct answer: d

The Stored Compiled Macro Facility enables you to store compiled macros permanently so that you can reuse them in later SAS sessions without compiling them again. Compiled macros must be stored in a catalog named *Sasmacr*, and both the MSTORED system option and the SASMSTORE= system option must be specified.

5. Correct answer: b

In order to create a permanently stored compiled macro, you must specify the MSTORED system option. The SASMSTORE= system option must be specified to point to the library in which you want your macros to be stored. You must also use the STORE option in the %MACRO statement.

6. Correct answer: a

When you submit a macro definition, SAS creates a session-compiled macro and stores it in the temporary SAS catalog *Work.Sasmacr*. This macro will be deleted at the end of the SAS session.

7. Correct answer: d

If you store your macro definitions in external files, you can easily share these files with others. Also, you can edit a macro definition that is stored in an external file with any text editor, and you can reuse the macro in other SAS sessions.

8. Correct answer: b

The PROC CATALOG step enables you to view a list of the contents of a SAS catalog. This might be especially useful if you store your macro definitions as SOURCE entries in permanent SAS catalogs. You might also use the PROC CATALOG step to see a list of the session-compiled macros that are stored in *Work.Sasmacr*.

9. Correct answer: c

In order to use the Stored Compiled Macro Facility, you need to specify the MSTORED and SASMSTORE= system options. The Stored Compiled Macro Facility saves the compiled macro in a permanent SAS catalog, but it does not save the macro definition. You cannot move a compiled macro across operating systems. Since you cannot re-create the macro definition from a compiled macro, it is a good idea to save your source program permanently as well.

10. Correct answer: a

The autocall macro facility stores macro definitions — not compiled macros — permanently. The first time an autocall macro is called during a SAS session, the macro is compiled and a session-compiled macro is created in *Work.Sasmacr*. You can have multiple autocall libraries that are concatenated, and you can use the autocall facility in conjunction with the Stored Compiled Macro Facility.